**JavaScript**

JavaScript is an object-oriented scripting language that is smaller than Java. Being a client-side language, it runs in the web browser on the client-side with a simplified set of commands, easier code and no need for compilation.

**Perl**

Being a high-level programming language, its emphasis lies in code readability and clear syntax. It combines Object-oriented and functional programming styles, and is often used as a scripting language. Perl is an open-source language used widely to process text through CGI programs.

**Python**

This is an event-driven programming language which is extensively used by Google because of its simplicity. It is managed by the Python SoftwareFoundation.

**(Visual) Basic**

This is an event-driven programming language which is implemented on Microsoft’s .Net framework.

**PHP**

This language is especially suited for Web development because of it easy embedding into HTML pages. It is an open-source, server-side, cross-platform, interpretive HTML scripting language

**Objective-C**

This object-oriented programming language created first by Brad Cox and Tom Love at their company Stepstone in the early 1980s, adds Smalltalk-like messaging to the C programming language.

**C++**

C++ is a general purpose multi-paradigm spanning compiled language that has both high-level and low-level languages’ features. It was started as an enhancement to the C programming language, Bjarne Stroustrup in 1979.

**C#**

This general-purpose programming language developed by Microsoft evolved from C and C++ as a part of the software company’s .NET initiative.

**C**

C, a general purpose programming language built by Dennis Ritchie when he was a part of Bell Telephone labs, is the bass of C++ and other programming languages. It was built to work with the Unix operating system.

**Java**

 An object-oriented programming language developed in the late 1990s by James Gosling and colleagues at Sun Microsystems.

Some notable features of C# that distinguish it from C and C++ (and Java) are:

* It has no global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.
* Local variables cannot shadow variables of the enclosing block, unlike C and C++.
* C# supports a strict Boolean data type, bool. Statements that take conditions, such as while and if, require an expression of a type that implements the true operator, such as the boolean type. While C++ also has a boolean type, it can be freely converted to and from integers, and expressions such as if(a) require only that a is convertible to bool, allowing a to be an int, or a pointer. C# disallows this "integer meaning true or false" approach, on the grounds that forcing programmers to use expressions that return exactly bool can prevent certain types of common programming mistakes in C or C++ such as if (a = b) (use of assignment = instead of equality ==).
* In C#, memory address pointers can only be used within blocks specifically marked as *unsafe*, and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined null value; it is impossible to obtain a reference to a "dead" object (one that has been garbage collected), or to a random block of memory. An unsafe pointer can point to an instance of a value-type, array, string, or a block of memory allocated on a stack. Code that is not marked as unsafe can still store and manipulate pointers through the System.IntPtr type, but it cannot dereference them.
* Managed memory cannot be explicitly freed; instead, it is automatically garbage collected. Garbage collection addresses the problem of memory leaks by freeing the programmer of responsibility for releasing memory that is no longer needed.
* In addition to the try...catch construct to handle exceptions, C# has a try...finally construct to guarantee execution of the code in the finally block, whether an exception occurs or not.
* Multiple inheritance is not supported, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI.
* C#, like C++, but unlike Java, supports operator overloading.
* C# is more type safe than C++. The only implicit conversions by default are those that are considered safe, such as widening of integers. This is enforced at compile-time, during JIT, and, in some cases, at runtime. No implicit conversions occur between booleans and integers, nor between enumeration members and integers (except for literal 0, which can be implicitly converted to any enumerated type). Any user-defined conversion must be explicitly marked as explicit or implicit, unlike C++ copy constructors and conversion operators, which are both implicit by default. Starting with version 4.0, C# supports a "dynamic" data type that enforces type checking at runtime only.
* Enumeration members are placed in their own scope.
* C# provides properties as syntactic sugar for a common pattern in which a pair of methods, accessor (getter) and mutator (setter) encapsulate operations on a single attribute of a class. No redundant method signatures for the getter/setter implementations need be written, and the property may be accessed using attribute syntax rather than more verbose method calls.
* Checked exceptions are not present in C# (in contrast to Java). This has been a conscious decision based on the issues of scalability and versionability.
* Though primarily an imperative language, since C# 3.0 it supports functional programming techniques through first-class function objects and lambda expressions.